What to expect from this tutorial?

You may feel a little clueless if you are coding for the first time in Python. This document has been prepared to get you all started with the first part of HW4. This serves as an introductory guide to a few specific concepts in Python. These may be used in order to complete Task1 in HW4. However, this cannot be deemed as a complete guide to learning Python. Look out for various hyperlinks for detailed explanation of concepts. Python version 2.7.11.

1. Python’s Core Data Types:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>5467, 9.326, 4+9j</td>
</tr>
<tr>
<td>Strings</td>
<td>'Alice','Bob's', b'a\x01c', u'sp\xc4m'</td>
</tr>
<tr>
<td>Lists</td>
<td>[1,2,3,4], [4,[9,'eggs'], 7.8,322]</td>
</tr>
<tr>
<td>Dictionaries</td>
<td>{&quot;a&quot;:1,&quot;b&quot;:2,&quot;c&quot;:3}</td>
</tr>
<tr>
<td>Tuples</td>
<td>(6,&quot;Bob&quot;,8,&quot;F&quot;)</td>
</tr>
<tr>
<td>Sets</td>
<td>set('xyz'), {x',y',z'}</td>
</tr>
<tr>
<td>Program unit types</td>
<td>Functions, modules, classes</td>
</tr>
</tbody>
</table>

2. Conditional Statements:

Conditional statements are used to perform different actions depending on whether the statement is True or False. Refer to this for more information.

3. For Loops:

The for statement acts as an iterator in Python. The for loops work on strings, lists, tuples, built-in iterables and user defined objects (later discussed). Syntax is as follows:

```python
for target in object:
    statements  #Assign object items to target
    #Repeated loop body
else:
    statements  #Optional else part
```

When python runs for loop, it assigns the iterable item in object to the variable target one by one and executes the loop body for each. The for statement also supports an else block.
Conditional statements in for loop:

```python
for target in object:  #Assign object items to target
    statements
    if test: break      #Exit loop now, skip else
    if test: continue   #Go to top of loop now
else:
    statements         #if ‘break’ wasn’t hit
```

4. While loops:

While statement is one of the most common iterator construct in Python. It repeatedly executed a block of code till the test specified on top turns in to be true. When the test statement turns false, the block of code that follows the while block is executed. Syntax:

```python
while test: #loop test
    statements #code block executed if test true
else:
    statements #optional else
    #run if didn’t exit loop with break
```

5. break, continue, pass:

Break and continue have a purpose only when nested inside loops. Pass is python’s empty placeholder statement and its usage is not necessarily tied to loops. Definitions are as follows:

```python
break:
    The code execution pointer jumps out of the closest execution loop.
continue:
    The code execution pointer jumps to the top of the closest execution loops.
pass:
    Does nothing. Empty statement placeholder.
```

6. String Fundamentals:

This portion of the tutorial only covers a very little part of the string story in Python. It consists of the part that may be mostly used in the scripts required for the homework. To get the complete story, refer here.

Strings are sequences of one-character strings. Remember that string supports operations that assume positional ordering among items. Also indexing in strings starts with 0 i.e, the first item is indexed at 0.
Indexing in strings:

>>> S='visual'
>>> len(S) #length of the string
6
>>> S[3]
'v'
Backward indexing in strings:

>>> S[-1]
'l'

Strings are immutable. As you will observe, every string operation is used to produce a new portion (partial/whole) of the string, since strings are immutable in python. By immutable it means that any item in a string cannot be changed once it has been created.

>>> S
'visual'
>>> S[0]= 'f'
TypeError: 'str' object does not support item assignment

List of useful string operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S[i]</td>
<td>Indexing</td>
</tr>
<tr>
<td>S[i:j]</td>
<td>Slicing</td>
</tr>
<tr>
<td>len(S)</td>
<td>Length of the string</td>
</tr>
<tr>
<td>S.find('isu')</td>
<td>Search</td>
</tr>
<tr>
<td>S.rstrip()</td>
<td>Remove white space</td>
</tr>
<tr>
<td>S.replace('isu','xxx')</td>
<td>Replacement</td>
</tr>
<tr>
<td>S.split(',')</td>
<td>Split on delimiter</td>
</tr>
<tr>
<td>S.lower()</td>
<td>Case conversion</td>
</tr>
</tbody>
</table>

You may visit [this](#) website, to practice upon various basic string operations.
Help!

Many more methods are available for string objects. In order to find the list of all attributes (methods are function attributes) you may use the built in `dir` function.

```python
>>> dir(S)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
 '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__',
 '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
 '__str__', '__subclassook__', 'capitalize', 'casefold', 'center', 'count',
 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index',
 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower',
 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex',
 'rjust', 'rpartition', 'rsplit', 'rsplit', 'split', 'splitlines', 'startswith',
 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

7. Functions:

So why do we need to use functions afterall? You may wish to use a certain logic which may afterall be more than one line of code more than one time in your python script. So functions help your package your logic into a more generalized code and reuse it as required. Also function lets you split your code into various pieces where each piece may be associated with a certain work as you will see in the starter code provided to you for the task. Python functions are written with the statement `def`. Also remember that def is an executable statement, meaning that the function does not actually gets created (like in compiled languages C etc.) until python reaches and runs it. The `return` statement which you again may see in the code result object back to the caller. This `return` statement may actually appear anywhere in the function. This when reached return an optional object (can be empty too) back to the place where it was called from.

Syntax:

```python
def func1(arg1, arg2, agr3...):
    ...
    return value
```

The function `func1` is passed a list of zero or more arguments list which will be utilized while executing the clock of code enclosed by it. The function will finally return a `value` (or return nothing). Technically, a function without a `return` return a None object to the caller.

How do you “call” this function?

You can simply call it by adding parentheses after the function name and adding the list of arguments within them.
You may practice on functions [here](#).

8. Classes:

To understand this section can be a little overwhelming especially if you are new to object oriented programming. We will try to explain it as simply as possible. Classes are python’s main OOP tools.

Earlier in this tutorial we saw python’s core data types. These are what are called “built-in” object types. However, we could also create our very own objects using classes! So why do we need to create new objects? We may need a one place package which actually lets us define a lot of built-in data types like numbers, string, lists and also lets us define functions in them. Classes are basically factories for generating one or more objects. Every time we call a class, we generate a new object with a distinct namespace. Each object thus created has access to all the attributes of the class and has its own namespace to store its data.

**Classes generate instance objects:**

Classes generate instance objects when called upon. Instance objects can be understood as a namespace which has access to all the attributes of the class (i.e., built in data types enclosed, functions defined within the class). Each instance has its own independent data. Every time an instance gets created (upon calling the class), it gets its own memory space but inherits the attributes of the class.

**What is this self in the starter code?**

You may have noticed the word self which appears in the first argument in all the method functions within the class RandomForest. This first argument called self by convention references the instance object being processed. Let us understand all of this through an example:

```python
>>>class Hello: #Define a class object
    def assignment(self, value): #Define class’s methods
        self.data=value #self is the instance
    def display(self):
        print(self.data)

>>>x=Hello()
```

By calling the class this way, an instance object x is generated which has its own namespace and access to all the attributes of the class. In the assignment function within the class, the
value passed in is assigned to self.data. Now the self (leftmost argument) automatically refers to the instance being processed (in this case x) and stores the value in this instance’s namespace.

Methods named with double underscores (__X__):

In python classes, operator-overloading is implemented using specially named methods to intercept operators. __init__ method is run when a new instance object is created.

9. NumPy:

The starter code also makes use of NumPy. To see what it does, refer here. The two numpy functions used in the starter code are argmax and bincount.

10. from, import:

A module is a file containing python definitions and statements. The filename is the module name with suffix .py appended to it. So in the starter code:

```python
from CSE6242HW4Tester import generateSubmissionFile
```

CSE6242HW4Tester is the module name used in RandomForest.py. Modules can import different items in this case function generateSubmissionFile.